Towards a Better Understanding of Software Product Line Evolution

Christian Kröher, Klaus Schmid
University of Hildesheim, Institute of Computer Science
Universitätsplatz 1, 31141 Hildesheim
{kroeher, schmid}@sse.uni-hildesheim.de

Abstract

In contrast to traditional software systems, the evolution of a Software Product Line (SPL) affects not only artifacts like source code or requirements, but also variability information, which supports the customization of these artifacts across different products of the SPL. While some work exists that aims at characterizing the state and evolution of a product line from a feature perspective, this abstracts away the details of code evolution, hence, ignoring aspects like the difference in size of features. In this paper, we present an approach for the extraction and analysis of changes introduced to code, build, and variability model artifacts. This approach has been developed for and applied to the Linux product line.

1 Introduction

Traditional software systems consist of different types of information like program and build process definitions contained in code and build artifacts, respectively. In a Software Product Line (SPL) these artifacts are extended by variability information in order to develop generic artifacts that can be reused in different products [4]. These generalized artifacts thus provide configuration options that allow the adaptation of their capabilities for a specific product. For example, in C-code this is typically done by augmenting the implementation with pre-processor statements. Besides these generalized artifacts, typically a SPL-specific artifact exists: the variability model, which is an abstract representation of the often complex and distributed configuration options and their relations [4].

The evolution of SPLs thus affects not only artifact-specific information as in traditional software systems, but also variability information. Hence, a particular focus of research in the area of SPLs is on understanding the evolution of variability information [1, 3, 2]. However, to the best of our knowledge, there is so far no research on how artifact-specific and variability information in different types of artifacts co-evolve in real-world, mature SPLs over time. In particular, the distribution and intensity of changes to the different types of information and artifacts are unknown. We argue that such information contributes to the understanding of SPL evolution at large and provides a basis for future work on supporting SPL evolution.

This paper describes an approach to extract and analyze the evolution of the Linux kernel (and SPLs relying on the same technologies), which is one of the largest open-source SPLs. The approach analyzes 662.110 commits of the Linux kernel repository spanning 11 years of active development. The focus of this analysis is on the largest groups of artifacts: source code, build, and variability model artifacts. The result is an evolution data set containing commit-wise numbers on the changed artifact types and, for each artifact type, the numbers on changed lines containing artifact-specific information and changed lines containing variability information.

2 Approach

The evolution analysis consists of two distinct processes: the Commit Extraction (ComEx) and the Commit Analysis (ComAn) process. As the repository under analysis relies on the git version control system, the ComEx process executes a set of git commands to extract the commit history in terms of individual files for each commit. The first line of such a commit file contains the commit date, while the following lines describe the changes to individual files. Change descriptions always start with a git diff-header defining basic properties of a changed file, like the relative path of that file or the number of lines before and after the change. After that header, the full content of the changed file is listed. Leading "+" (addition) or "-" (deletion) symbols mark changes to individual lines of that content.

The ComAn process investigates each commit file to identify changes to the artifacts in the focus of the approach and to count the number of changed lines containing artifact-specific information and variability information for each of these artifacts separately. The first step is therefore to determine the type of the changed file by checking the name and file extension. In the Linux kernel source code artifacts are identified by an ".h", ".c", or ".S" file extension, while build artifacts are named "Makefile" or "Kbuild". Variability model artifacts are named "Kconfig". For each

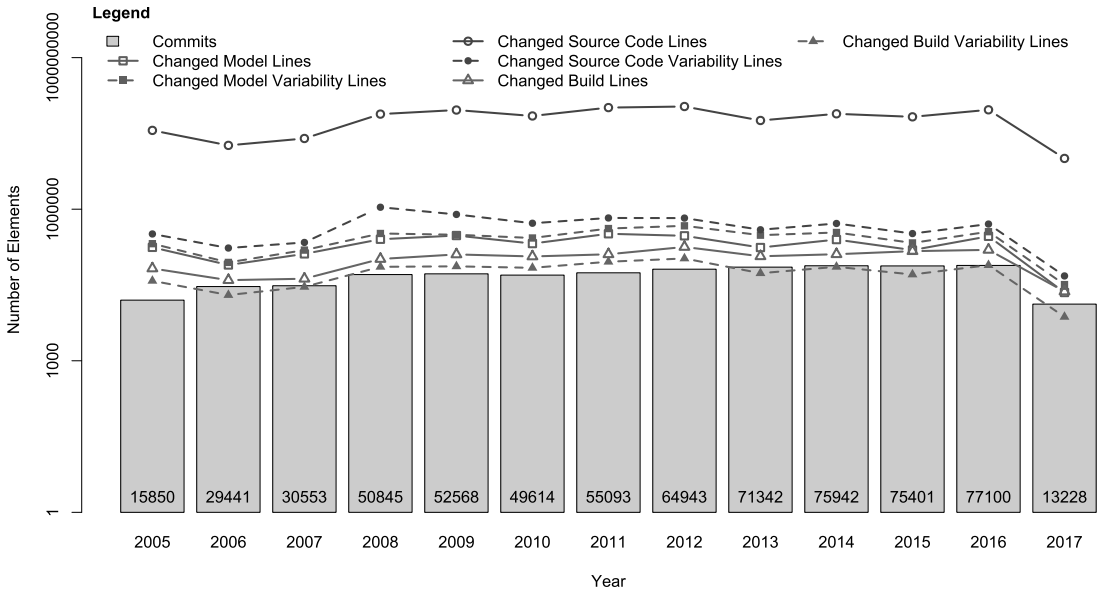

Figure 1: Evolution of the Linux kernel

type of artifact the ComAn process performs a specialized analysis. This analysis ignores changes to empty lines and comments as they neither constitute artifact-specific capabilities nor variability information. The remaining lines are categorized as follows:

Source code and build artifacts: Changed lines, that either contain references to configuration options or statements closing a control structure, which depends on a configuration option, are counted as changes to variability information. All other changed lines are counted as changes to artifact-specific information.

Variability model artifacts: In contrast to the other artifact types, the focus is on finding artifact-specific information instead of variability information. This is due to the fact that for a variability model the artifact-specific information is variability information. However, some information, like help texts for configuration options, does not influence the variability per se. Thus, changes to help texts are counted as changes to artifact-specific information, while all other changes are counted as variability information.

3 Results

The numbers provided by the analysis enable different types of evaluations. Figure 1 shows one example of these evaluations: an overview of the evolution of the Linux kernel by summarizing the numbers of commits and changed lines by year. For each year, the bar illustrates the number of commits including the actual number. The lines visualize the number of changes to different types of artifacts, like the solid line with empty circles representing the number of changed source code lines without variability impact. This is the largest set of changes. Variability information in code artifacts changes significantly less, while these changes occur more often than to the build system or to variability model artifacts. By far not all changes to the variability model impact the variability, which may appear surprising. There are about as many changes to the variability model with variability impact as there are without. The latter ones typically make descriptive changes, e.g., modifying help information. The build artifacts change the least with more relating to non-variability-related information than to variability-related information.

The results of our evolution analysis show that a significant amount of changes apply to variability information regardless of the type of artifact. In the future we will extend this analysis to further SPLs in order to compare the results with those derived from Linux. Our goal is to draw more general conclusions about the evolution of SPLs and to provide them as input for future work on supporting SPL evolution.

Acknowledgment

This work was partially supported by the DFG under Priority Program SPP1593 and by the BMBF, FKZ: 01S16042H: project REVAMP^2, ITEA3-15010.

References

[1] N. Dintzner, A. van Deursen, and M. Pinzger. FEVER: Extracting feature-oriented changes from commits. In *Conference on Mining Software Repositories*, pages 85–96, 2016.

[2] R. Lotufo, S. She, T. Berger, K. Czarnecki, and A. Wasowski. Evolution of the linux kernel variability model. In *Intern. Software Product Line Conference*, pages 136–150, 2014.

[3] L. Passos, J. Guo, L. Teixeira, K. Czarnecki, A. Wasowski, and P. Borba. Coevolution of variability models and related artifacts: A case study from the linux kernel. In *Intern. Software Product Line Conference*, pages 91–100, 2013.

[4] K. Schmid and E. S. Almeida. Product line engineering. *IEEE Software*, 30:24–30, 2013.